

# Parallel Marker-Based Image Segmentation with Watershed Transformation<sup>1</sup>

Alina N. Moga

*Albert-Ludwigs-Universität, Institute for Informatics,  
Universität Gelände Flugplatz, Geb. 052, D-79085 Freiburg i.Br., Germany*

and

Moncef Gabbouj

*Signal Processing Laboratory, Tampere University of Technology,  
P.O. Box 553, FIN-33101 Tampere, Finland*

Received February 19, 1997; revised December 13, 1997; accepted January 10, 1997

---

The parallel watershed transformation used in gray-scale image segmentation is here augmented to perform with the aid of a priori supplied image cues called markers. The reason for introducing markers is to calibrate a resilient algorithm to oversegmentation. In a hybrid fashion, pixels are first clustered based on spatial proximity and gray-level homogeneity with the watershed transformation. Boundary-based region merging is then effected to condense nonmarked regions into marked catchment basins. The agglomeration strategy works with a weighted neighborhood graph representation of the oversegmented image. The throughput of a parallel Borůvka-like minimum spanning forest (MSF) operator, applied on the considered graph, embodies the desired image partition, reasoning that all regions in a tree fuse into a homogeneous area containing a unique marker. Two figures of merit of the parallel algorithm are worth mentioning: the local detection of the catchment basins conforming to the watershed principle (which strongly depends on the history of the regions' growth) and the parallel computation of the Borůvka-like MSF which merges, at the same time, partial regions, produced by the local labeling, and nonmarked regions to marked basins. Both modules are designed with great concurrency, locality, and reduced software engineering cost, emerging into a scalable algorithm. © 1998 Academic Press

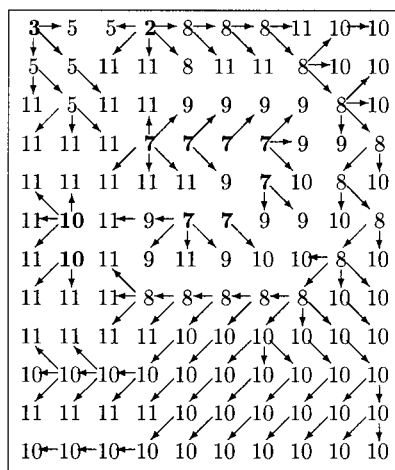
---

<sup>1</sup> This research has been supported by the Graduate School in Electronics, Telecommunication, and Automation (GETA), Finland, and the Edinburgh Parallel Computing Centre in the Training and Research on Advanced Computing Systems (TRACS) program.

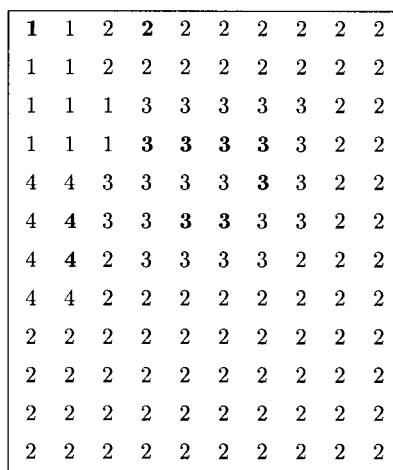
## 1. INTRODUCTION

Watershed transformation [2, 10, 11] is an effective tool for morphological image segmentation. The sequential algorithm starts by detecting and labeling initial seeds, e.g., regional minima of the gradient (connected plateaus of constant altitude which do not have neighboring pixels of a lower gray-level). Starting from minima, ordered region growth is then performed. Thus, nonlabeled pixels are assimilated into different components in increasing order of gray-levels. Inside flat areas which are not yet labeled, called plateaus of nonminima, components progress synchronously, such that they incorporate equal extents within the plateau. Consequently, a pixel on such a plateau is labeled along the shortest path, completely included in the plateau, to a lower brim of the plateau. The set of nonoverlapping connected components is a complete partition of the image and their labels depict the output image. Flooding in a simple input image and its resulting labels are illustrated in Fig. 1.

When applied to real images, the transformation produces oversegmentation. A solution to cope with this problem was to mark meaningful parts of the original image and to constrain the segmentation process to grow only one region around each marker. Two approaches are feasible: first, to modify the homotopy of the gradient image [2], in order to obtrude regional minima only at the selected marker locations, and then to apply the classical watershed algorithm; and second, a post-processing of the oversegmented watershed image, which merges every nonmarked region to the nearest marked one, according to a given distance metric. Suppose that in Fig. 1b pixels (3, 3), (4, 6), and (8, 6) (*line, col*) coordinates are relative to the (0, 0) top left corner of the image) are marked; then regions 1 and 4 have to be appended to a marked area, labeled 2 and 3. Let us first remark that merging is performed regionwise, and second that a catchment basin which contains more than one marked pixel is not further split (see region labeled 3 in Fig. 1b), but will not merge with any other marked basin. The solution for region merging originates in [12].



a



b

FIG. 1. Sequential watershed: (a) flooding the input image; (b) output image of labels.

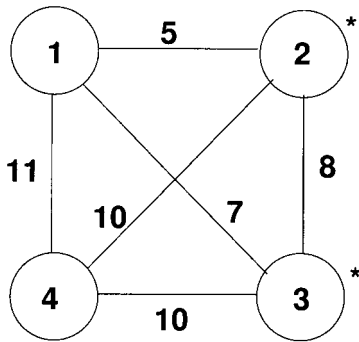


FIG. 2. The WNG for the image in Fig. 1b.

Thus, a weighted neighborhood graph is first constructed with vertices representing the basins in the oversegmented image, and edges the neighborhood relation between regions. The weight of an edge is set to the lowest gray-level pass between the two neighboring regions. The graph of the image in Fig. 1b can be observed in Fig. 2 (8-connectivity has been used). Further on, starting from marked vertices, flooding along the lightest outgoing edge toward an unmarked node is performed. Thus, node 2 appends node 1 along the edge (2, 1) of weight 5, and node 3 floods node 4 along the edge (3, 4) of weight 10; edges (3, 1) of weight 7 and (3, 2) of weight 8 are not considered since they connect marked components (note that node 4 could also be appended to node 2, having the same distance 10 as to node 3). The same result is obtained if the minimum spanning forest (MSF) of the graph is computed, constraining every tree in the MSF to contain exactly one marked vertex (region) (see [12]).

For the first part of the problem, namely, parallelization of the watershed algorithm, an efficient connected-components algorithm has been presented in [18]. In this paper, the algorithm is modified to additionally perform in parallel marker-based region merging by means of a constrained MSF operator. The algorithm represents an extension of the classical MSF problem, namely, to find the MSF of a graph with vertices marked, in part, such that in every tree, exactly one node is marked.

In the next section, region merging is first theoretically formalized as a constrained graph-partition problem and then reduced to a marker-conditioned MSF computation. Section 3 describes the parallel marker-based watershed transformation with emphasis on marker-based region merging. Next, a theoretical complexity analysis of the parallel algorithm is developed in Section 4. The performance model is reassessed on an experimental basis in Section 5. Additionally, this section illustrates the results of the image segmentation process. Finally, Section 6 concludes the paper.

## 2. REGION MERGING AS A CONSTRAINED MSF PROBLEM

Let  $WNG = (V, E)$  denote an undirected weighted neighborhood graph. The vertices in  $V$  correspond to catchment basins in the oversegmented image. The

edges in  $E$  are  $e = (u, v)$ , such that the regions  $u$  and  $v$  share a common boundary. The weight of  $e$ ,  $w(e)$ , is equal to the lowest graylevel along the common interface line between  $u$  and  $v$ . For any two nodes which are not linked by an edge, the weight is considered to be  $\infty$ . In addition, a subset  $M \subseteq V$ ,  $M = \{m_i\}$ , represents the marked nodes. The region merging problem can be stated now as finding a partition of  $WNG$  consisting of  $|M|$  connected components  $WNG_i = (V_i, E_i)$ ,  $0 \leq i < |M|$ ,

$$\bigcup_{i=0}^{|M|-1} V_i = V \quad \text{and} \quad V_i \cap V_j = \emptyset, \quad i \neq j \text{ such that} \quad (1)$$

$$\forall i, V_i \cap M = \{m_i\} \text{ (any component contains exactly one marked vertex)} \quad (2)$$

and

$$\forall u \in V_i, \mathcal{D}(u, m_i) = \min_{m_j \in M} \{\mathcal{D}(u, m_j)\} \text{ (} u \text{ merges the closest marked node)}. \quad (3)$$

$\mathcal{D}(u, v)$  measures the distance between two nodes  $u, v$ , and it is given by the length of the shortest path connecting the two nodes:

$$\mathcal{D}(u, v) = \min_{\gamma \in \Gamma(u, v)} \mathcal{L}(\gamma). \quad (4)$$

$\Gamma(u, v)$  is the set of all existing paths  $\gamma$  between  $u$  and  $v$ . The length of a path  $\mathcal{L}(\gamma)$  is defined as the weight of the heaviest edge in the path:

$$\mathcal{L}(\gamma) = \max_{e \in \gamma} \{w(e)\}. \quad (5)$$

Partitioning the weighted neighborhood graph as above is equivalent to flooding according to watershed principle, which progresses through the lowest pass. Proving that the shortest paths finding problem is tantamount to computation of MSF, i.e., of a set of spanning trees of  $WNG$  whose total edge cost is minimum, is not intricate.

**THEOREM 1.** *An edge  $e = (u, v)$  is in the MSF iff  $w(e) = \mathcal{D}(u, v)$ .*

*Proof.* For the sake of simplicity, suppose that every marked node is linked to an artificial node with an edge lighter than any existing edge in the graph and the minimum spanning tree (MST) of the new graph is computed. Every artificial edge is definitely an edge in the MST. Otherwise, by including it the MST, a cycle is formed. Removing any edge in this cycle, heavier than the last introduced edge, a lower cost MST is obtained: contradiction. The MSF of the initial graph can be thus yielded by trimming all artificial edges from the MST of the extended graph. Therefore, in the next demonstration, the MST will be used.

" $\Rightarrow$ " If  $e = (u, v)$  is in the MST then  $w(e) = \mathcal{D}(u, v)$ . Supposing  $w(e) \neq \mathcal{D}(u, v) \rightarrow \exists \gamma' \in \Gamma(u, v)$ ,  $\gamma' \neq \gamma = \{e\}$ , such that  $\mathcal{L}(\gamma') = \mathcal{D}(u, v)$ . From Eq. (4)  $\rightarrow w(e) = \mathcal{L}(\gamma) > \mathcal{D}(u, v) \rightarrow \forall e' \in \gamma'$ ,  $w(e') \leq \mathcal{D}(u, v) < w(e) \rightarrow$  in the cycle  $(\gamma' \cup \{e\})$   $e$  is

the heaviest edge. Replacing  $e$  with any edge from  $\gamma'$  which passes the cut which  $e$  crosses, a spanning tree with lower cost is obtained: contradiction with  $e$  in MST.

“ $\Leftarrow$ ” If  $w(e) = \mathcal{L}(u, v)$  then  $e$  is in MST. Suppose  $e$  is not an edge in the MST  $\rightarrow \exists \gamma' \in \Gamma(u, v)$  included in the MST. From Eq. (4)  $\rightarrow w(e) = \mathcal{L}(\gamma) < \mathcal{L}(\gamma')$ , where  $\gamma = \{e\}$ . Moreover, from Eq. (5)  $\rightarrow \exists e' \in \gamma'$ ,  $w(e') = \mathcal{L}(\gamma') \rightarrow w(e) < w(e')$ . Replacing in the cycle  $\gamma' \cup \{e\}$   $e'$  with  $e$ , a lower cost spanning tree is yielded, a contradiction.

The problem of merging regions is thus reduced to computation of the MSF with the constraint that *any tree of the forest contains exactly one marked node*. In the next section, parallelization of the whole algorithm is described.

### 3. THE PARALLEL MARKER BASED WATERSHED ALGORITHM

#### 3.1. Problem Decomposition

The current approach uses the *divide-and-conquer* paradigm with regular domain decomposition. Thus, the global image of size  $l \times l$  is uniformly split into blocks of approximately  $(l/\sqrt{P}) \times (l/\sqrt{P})$  and each block (subimage) is mapped on one processor of a  $P$  processors logical grid. Because operations on each processor are neighborhood oriented, communication is often needed for pixels on edges and corners of a subimage. In order to minimize the communication cost and increase the locality, edge/corner pixels are replicated in the neighboring processors, whose subdomains are thus enlarged with a haul of one line/column/point.

A good partition, however, divides both data and computation associated with the problem. The idea is to exploit the local data and produce partial results which are then merged hierarchically tree- or pyramidwise into a global result in  $\log P$  steps. Still, the method requires that the operator used for each intermediate computation be associative and without dependencies (the order in which the partial results are combined is unimportant). This procedure is followed in our paper.

The parallel marker-based watershed algorithm consists of two parts. First, labeling according to the watershed principle to delimit the catchment basins in the oversegmented image; and second, merging the nonmarked regions to the closest marked area. Ostensibly, the watershed transformation does not possess much parallel potential, having a sequential, recursive nature. Different techniques have been used in [13–16] to produce a scalable, efficient parallel algorithm. However, another strategy of labeling connected components in parallel as in [1] was proven applicable for the watershed transformation which does not build watershed lines (0-width watershed lines) [18]. The main idea in [18] is to derive a local precedence relation between a candidate pixel and its already flooded neighbor, at which the candidate pixel will be appended. Once this relation is established, connected components are first locally correctly labeled. However, components which extend over more than one image subdomain are internally fragmented. Following then the predecessor–successor pixel relation in every processor, neighboring components lying on adjacent subdomains are represented as vertices linked by an edge in a boundary connectivity graph (BCG). Final components are obtained by a

global connected-components (GCC) operator which incorporates into a pyramidal hierarchy of masters and slaves the BCG from each processor (see Fig. 3a). Similarly, a global constrained MSF (GMSF\*) operator can be implemented for the watershed image  $O$ , when markers in  $M$  are known regionwise— $M[p] = 1$  if the region labeled  $O[p]$  is marked (see Fig. 3b). In this paper, a more efficient design solution applies the two global operators at the same time on the partial image of labels  $o$  and markers  $m$ — $m[p] = 1$  if  $p$  is a marked pixel (see Fig. 3c). Since the emphasis in this paper is more on parallelization of the marker based region merging problem, we do not detail further the local watershed labeling ( $wl$ ) (see also [18, 19]). However, for the purpose of marker-based region merging, the local labeling result ( $o$ ), the markers ( $m$ ), the BCG ( $bcg$ ), and the WNG ( $wng$ ) suffice, and the explanation follows next.

A BCG can also be regarded as a WNG in which all edges have the same arbitrarily chosen weight. Thus, computing the connected components of a BCG can be solved by an MSF operator. Each tree in the forest represents a connected component. Recall that the WNG, whose MSF groups regions to be merged in order to reduce the oversegmentation, contains interregion links, while BCG stores

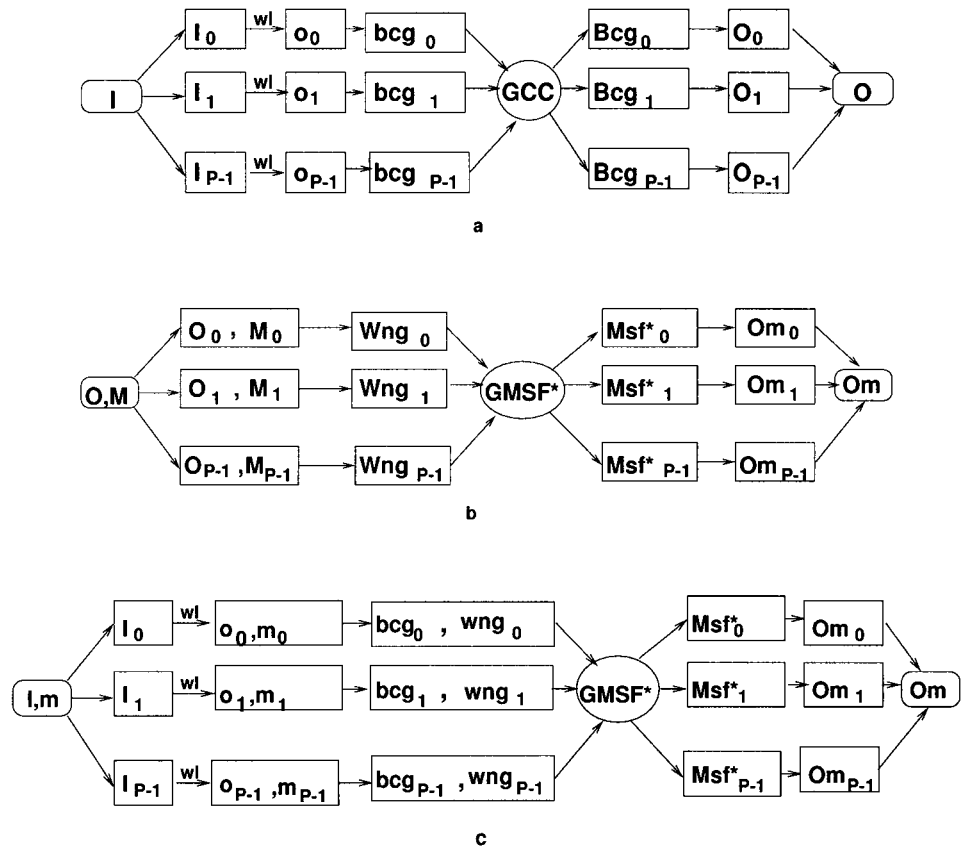
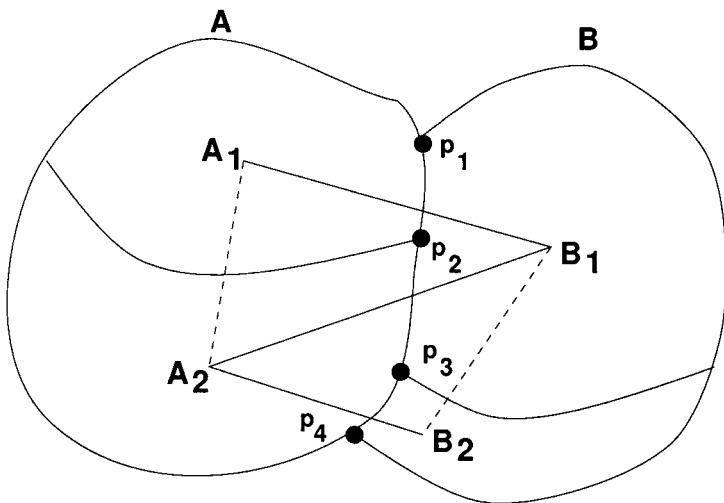


FIG. 3. Parallel design solution for (a) watershed, (b) marker-based region merging, and (c) marker-based watershed. Variables starting in a capital letter represent global data and in lower-case partial data.



**FIG. 4.** Intra- (dashed line) and interregion links (solid line) between two global catchment basins,  $A$  and  $B$ .

intraregion binds (where intra- and inter- refer to *global* catchment basins produced by watersheds). In Fig. 4, two global regions  $A$  and  $B$  consist each of two fragments  $A_1$ ,  $A_2$  and  $B_1$ ,  $B_2$ , respectively. The intraregion links in BCG are shown in dashed line and the interregion links in WNG in solid line. The two graphs are, however, edge disjoint. The weight of  $e(A_1, B_1)$ ,  $e(A_2, B_1)$ , and  $e(A_2, B_2)$  is equal to the minimum graylevel along the contour  $(p_1, p_2)$ ,  $(p_2, p_3)$ , and  $(p_3, p_4)$ , respectively. If the weights of the intraregion edges  $w(e(A_1, A_2))$ ,  $w(e(B_1, B_2))$  are lighter than any weight of the interregion links  $w(e(A_1, B_1))$ ,  $w(e(A_2, B_1))$ ,  $w(e(A_2, B_2))$ , then the MSF of  $BCG \cup WNG$  will first select the edges of minimum weight,  $e(A_1, A_2)$  and  $e(B_1, B_2)$  (which corresponds to condensing  $A_1$  with  $A_2$  and  $B_1$  with  $B_2$ ), and then the lightest edge between the interregion links. The latter has actually the cost equal to the minimum gray-level along the whole contour between  $p_1$  and  $p_4$ . In conclusion, by embedding the equal cost edges of the BCG into the WNG, so that the weight of the edges in BCG are the lightest in the resulting graph—for example  $-1$ , provided that all weights in WNG are greater than or equal to  $0$ —and by computing the MSF of the emerged graph, the same result is obtained as if the MSF of the WNG of the global regions would have been computed. Consequently, the solution in Fig. 3c is applicable.

### 3.2. Description of the Parallel Algorithm

As emphasized in Section 2, the marker-based region merging problem can be solved by an MSF operator, with the constraint that *any tree in the forest contains exactly one marked vertex*. There are three classical algorithms for generating a MSF [6, 22]: Borůvka (rediscovered by Choquet, Lukaszewicz, and Sollin), Kruskal, and Jarník (rediscovered by Prim and Dijkstra). All algorithms start with a set of trivial fragments and selectively add edges until it becomes an MSF. The difference occurs in the criterion used to select the next edge to be added at

each iteration. The algorithm which exploits best the data locality is Borůvka algorithm; it does not require global order and access of data at each iteration. Therefore, the optimal round robin version of the algorithm [22] is here parallelized and analyzed.

A holistic description of the sequential method is to start from a forest of one-node trees and grow minimum spanning trees along the lightest edge adjacent to the tree. The procedure stops when no more outgoing edges are left unvisited. The foremost important feature of the algorithm is the lack of order in expanding trees. Indeed, this property envisages a parallel implementation conforming the four attributes mentioned in the abstract, namely concurrency, locality, reduced software engineering cost, and scalability. In a *divide-and-conquer* manner, trees are grown locally in each processor until they become isolated or their branches need to pervade neighboring subdomains. By message-passing, in a pyramidal hierarchy of master-slave processors, parts of the same tree astride a common subdomain boundary coalesce or split, in the same manner as in the local growth. In the processor at the apex of the pyramid, the global MSF will be available. Several parallel implementations of the Borůvka algorithm exist in the literature [5, 7, 9, 21], but for our problem, we need a constrained parallel MSF operator. Moreover, the operator applies to a graph ( $wng$ ) of partial regions ( $o$ ) and with partial marking property ( $m$ ).

Let us resume the algorithm from the state it was left by the local flooding. Catchment basins were detected in  $o$ , but with intrinsic internal fragmentation due to domain decomposition. Furthermore,  $bcg$  yields in every processor the equivalence between labels assigned to parts of the same global catchment basin. For the image example in Fig. 1a, the subimages of labels  $o$  after the local watershed are illustrated in Fig. 5. The BCG for every processor in 8-connectivity is  $bcg_0 = \{[2, 5], [0, 5], [1, 8], [0, 9], [0, 7], [0, 11], [4, 6]\}$ ,  $bcg_1 = \{[6, 4], [5, 2], [5, 0], [5, 12], [6, 13]\}$ ,  $bcg_2 = \{[9, 0], [7, 0], [10, 13], [8, 1]\}$ ,  $bcg_3 = \{[11, 0], [12, 5], [13, 6], [13, 10]\}$ . The local WNG,  $wng$ , is locally computed in each processor, as described in Section 2. Once all inter- and intraregion edges ( $u, v, w$ ) ( $w$  is the weight of the edge ( $u, v$ )) have been collected in each processor in a new weighted graph, the latter is condensed such that every remaining edge satisfies the constraint that *either its weight is  $-1$  (compulsory intraregion link), or at most one of the two end vertices is marked (all interregion binds between two marked regions are discarded)*. At this stage, every processor encapsulates a weighted undirected graph in which two types of edges are distinguishable: edges with both ends internal nodes (labeling regions from the current processor) and edges with one external end (labeling a region from an adjacent processor). The graph is represented by the array of its nodes and for each node all its incident edges are stored in a list. Note that an edge ( $u, v$ ) will occur in both the edges lists of  $u$  and  $v$ , respectively, unless  $u$  or  $v$  is an external node. In this case ( $u, v$ ) appears only in the edges list of the internal node, while the external node has in the current processor an empty list of outgoing edges. In each processor, an array  $root[L = \sum_{i=0}^{P_i-1} L_i]$  ( $L_i$  is the number of assigned labels in processor  $P_i$ ) keeps track in every entry  $j$  of the root of the tree in which the node  $j$  has been incorporated during the MSF computation, described below in the BorůvkaMSF\* procedure. Initially,  $root[j] = j$ ,  $0 \leq j < L$ .

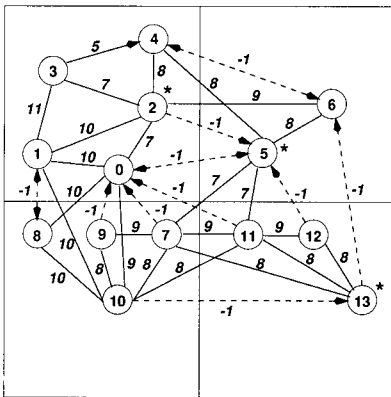
3	3	4	4	4	1	1	1	1	1
3	3	4	4	4	1	1	1	1	1
3	3	3	2	2	0	0	0	1	1
3	3	3	2	2	0	0	0	1	1
1	1	2	2	2	0	0	0	1	1
1	1	0	0	0	0	0	0	1	1
1	1	3	2	0	0	1	2	2	2
1	1	3	3	3	2	2	2	2	2
3	3	3	3	3	2	2	2	2	2
3	3	3	3	3	2	2	2	2	2
3	3	3	3	3	2	2	2	2	2
3	3	3	3	3	2	2	2	2	2

a

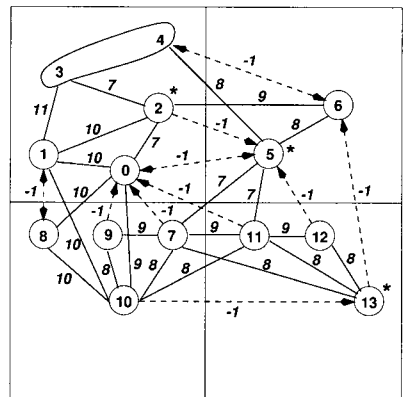
3	3	4	4	4	6	6	6	6	6
3	3	4	4	4	6	6	6	6	6
3	3	3	2	2	5	5	5	6	6
3	3	3	2	2	5	5	5	6	6
1	1	2	2	2	5	5	5	6	6
1	1	0	0	0	5	5	5	6	6
8	8	10	9	7	11	12	13	13	13
8	8	10	10	10	13	13	13	13	13
10	10	10	10	10	13	13	13	13	13
10	10	10	10	10	13	13	13	13	13
10	10	10	10	10	13	13	13	13	13
10	10	10	10	10	13	13	13	13	13

b

FIG. 5. Image of labels after the local watershed flooding: (a) relative labels and (b) absolute labels.



a



b

FIG. 6 (a) A weighted graph distributed to four processors and (b) the result of the local MSF operator applied in each processor.

In Fig. 6a,  $wng$  (with edges shown in solid line) and  $bcg$  (with edges drawn in dashed line and weight  $-1$ ) for the image in Fig. 5b, in 8-connectivity, are illustrated. Nodes having a  $\star$  attached represent marked regions. Note that there is no interregion link between the two marked nodes 5 and 13. The graph representation of processor  $P_0$  (top left), for example, is shown in the table below:

0	1	2	3	4	5	6	7	8	9	10	11	12	13
(0, 1, 10)	(1, 0, 10)	(2, 0, 7)	(3, 1, 11)	(4, 2, 8)									
(0, 2, 7)	(1, 2, 10)	(2, 1, 10)	(3, 2, 7)	(4, 3, 5)									
(0, 5, $-1$ )	(1, 3, 11)	(2, 3, 7)	(3, 4, 5)	(4, 5, 8)									
(0, 7, $-1$ )	(1, 8, $-1$ )	(2, 4, 8)		(4, 6, $-1$ )									
(0, 8, 10)	(1, 10, 10)	(2, 5, $-1$ )											
(0, 9, $-1$ )		(2, 6, 9)											
(0, 10, 9)													
(0, 11, $-1$ )													

*Initialization.* For each  $u \in V$ ,  $root[u] \leftarrow u$ ,  $unsolved[u] \leftarrow 0$ , if  $E[u] \neq 0$  then  $ADD(q_c, u)$ ;

*BorůvkaMSF\** ( $V, \mathcal{M}, E, root, q_u, unsolved, q_c$ )

while  $|q_c| > 1$  /\* nodes are processed in FIFO order \*/

$n \leftarrow GET(q_c)$ ;

(1) if  $E[n] = 0$  continue; /\*  $n$  has no outgoing edges  $\rightarrow n$  isolated node \*/

$(u, v, w) \leftarrow FindMin^*(E[n])$ ; /\* the lightest edge s.t.  $w = -1$  or  $(\mathcal{M}[find(root, u)]$  or  $\mathcal{M}[find(root, v)]) = 0$ ; if the weights are not distinct, ties are broken by ordering edges lexicographically after the end vertices \*/

if  $(u, v, w)$  does not exist then continue; /\*  $n$  turns out an isolated node \*/

(2) if  $\mathcal{M}[n] = 1$  and  $w \geq 0$  then /\* passive node; nonmarked adjacent nodes will hook on it, or its outgoing edges will be invalidated by appending their other ends to different marked components \*/

if  $unsolved[n] = 0$  then /\* store  $n$  as unsolved \*/

$unsolved[n] \leftarrow 1$ ,  $ADD(q_u, n)$ ;

(3) else  $r_u \leftarrow find(root, u)$ ,  $r_v \leftarrow find(root, v)$ ;

(3.1) if  $r_u = u$  and  $E[r_u] = 0$  then /\*  $r_u$  external node \*/

if  $unsolved[r_v] = 0$  then /\*  $r_v$  is unsolved \*/

$unsolved[r_v] \leftarrow 1$ ,  $ADD(q_u, r_v)$ ;

(3.2) else if  $r_v = v$  and  $E[r_v] = 0$  then /\*  $r_v$  external node \*/

if  $unsolved[r_u] = 0$  then /\*  $r_u$  is unsolved \*/

$unsolved[r_u] \leftarrow 1$ ,  $ADD(q_u, r_u)$ ;

(3.3) else /\*  $r_u, r_v$  are both internal nodes \*/

$r_{uv} \leftarrow union(root, r_u, r_v)$ ;  $E[r_{uv}] \leftarrow E[r_u] \cup E[r_v]$ ;

$\mathcal{M}[r_{uv}] \leftarrow \mathcal{M}[r_u]$  or  $\mathcal{M}[r_v]$ ;

if  $(unsolved[r_{uv}] \leftarrow (unsolved[r_u]$  or  $unsolved[r_v])) = 0$  then

$ADD(q_c, r_{uv})$ ;

*Remarks.*  $find(root, x)$  returns the root of the tree containing  $x$ , with compression of the path between  $x$  and its root.  $union(root, x, y)$  merges the disjoint trees

containing  $x$  and  $y$ , such that the root of the new tree is the minimum of the two component roots. Both functions are found in [4].

In Fig. 6a, for every node  $u$  the lightest incident edge  $(u, v, w)$  is depicted by an arrow from  $u$  to  $v$ . If the arrow surpasses the subdomain which encapsulates  $u$ ,  $u$  is stored as unsolved according to steps 3.1, 3.2. As may be observed, only in processor  $P_0$  is there an inner arrow from node 3 to node 4 which, conforming to step 3.3, coalesce.  $r_3 = \text{root}[3] = 3$ ,  $r_4 = \text{root}[4] = 4 \rightarrow r_{34} = 3$ , and  $\text{root}[4] = 3$ .

Because when two trees are merged the new edges list results by a simple concatenation of the two lists associated with the trees, *internal edges*  $(u, v)$ —having both end nodes in the same tree ( $\text{find}(\text{root}, u) = \text{find}(\text{root}, v)$ ), *invalid interregion edges*  $(u, v)$ —with both ends in marked trees ( $\mathcal{M}[\text{find}(\text{root}, u)] = 1$  and  $\mathcal{M}[\text{find}(\text{root}, v)] = 1$ ), and *multiple outgoing edges*  $(u_1, v_1), (u_2, v_2)$ —connecting the same two distinct trees ( $(\text{find}(\text{root}, u_1), \text{find}(\text{root}, v_1)) \Leftrightarrow (\text{find}(\text{root}, u_2), \text{find}(\text{root}, v_2))$ ) are not removed. This not only is memory cumbersome, but also considerably increases the complexity of the FindMin\* function. Therefore, trees are analyzed in the while loop in stages, such that in stage  $i$ , only trees of height  $i$  are considered for merging. One-node trees have height 1 and when two trees of height  $i$  and  $j$  are merged, the resulting tree has the height  $\min(i, j) + 1$ . After each stage, a cleanup is applied: internal and invalid edges are eliminated; multiple edges between distinct trees are all discarded, except the one with the minimal weight. However, multiple edges which have an external node are preserved. The motivation is given later in this subsection. If after cleanup all outgoing edges of a tree have been pruned, the tree becomes solved and it is dequeued.

The result of the MSF operator in each processor is shown in Fig. 6b. Thus, after cleanup, edge  $(3, 4, 5)$  has been removed, being an internal edge in the tree rooted at 3. Furthermore, edge  $(2, 4, 8)$  and edge  $(2, 3, 7)$  connect the same distinct trees rooted at 2 and 3. Consequently, the lighter edge,  $(2, 3, 7)$ , is retained in the graph, while edge  $(2, 4, 8)$  is elided. Nodes  $\{0, 1, 2, 3\}$  are unsolved in processor  $P_0$ , since their lightest edges are connected to external nodes. For the same reason, vertices  $\{5, 6\}$ ,  $\{7, 8, 9, 10\}$ , and  $\{11, 12, 13\}$  are unsolved in the processors they belong to.

An alternative approach would be to keep the edges list sorted and combine two lists not by simple concatenation, but by merge-sorting. The initialization of these lists would be thus more complex due to sorting, the combine routine would depend linearly on the lengths of both lists plus the time for *find* operations, albeit the selection of the minimum weight edge would take constant time. However, with the approach chosen here, initialization of the edges lists does not require any further processing, selection of the minimum weight edge depends only on the length of the edges list of the current node, and combination of two edges lists takes constant time. Since selection and combine are the most frequent operations, the time consumed in these routines is shorter with the current approach than preserving the edges lists ordered. In addition, after a cleanup step, all remaining trees are transformed into star graphs, which reduces the complexity of *find* calls in FindMin\* to  $O(1)$ .

Remark that when two trees are condensed, the weights of the remaining outgoing edges do not change. If the weights had measured a global feature dissimilarity between two regions linked by an edge, such as homogeneity, area, or region

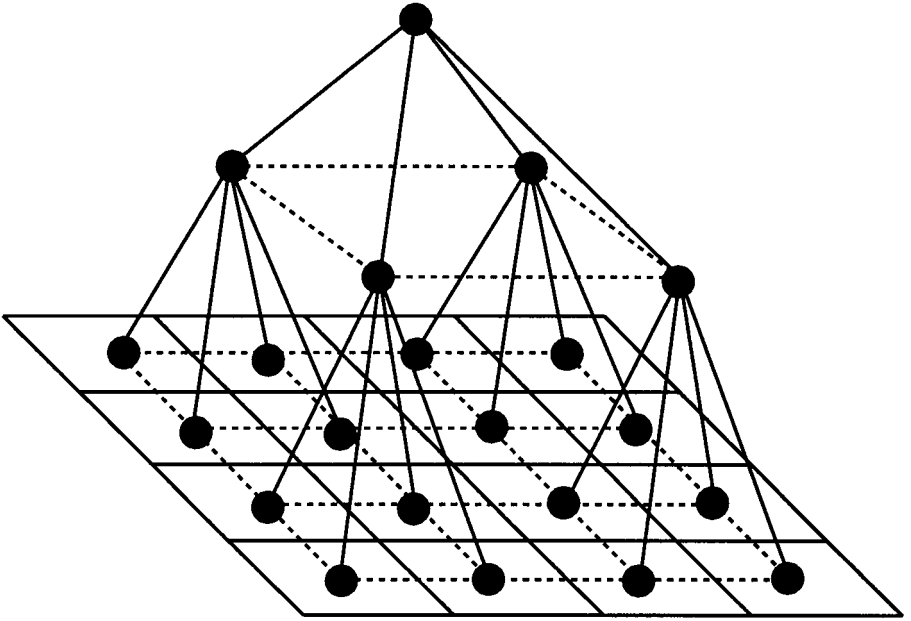


FIG. 7. The logical structure of master-slave processors to compute GMSF\* ( $\bullet$  is the local BorůvkaMSF\* operator).

histogram, they would have been affected by region merging. This is not, however, the case with the current definition of the weight. The encompassing contour of a merged region is the union of the boundaries of the component regions, except their common frontier; but the latter affects only the weight of the condensed edge.

After the BorůvkaMSF\* procedure terminates, each processor passes on to its master the unsolved nodes in  $q_u$ , after these are cleaned up. These nodes are representatives of components which either extend over the processor's boundaries or are marked and still have outgoing edges. For each such node, its marker and list of outgoing edges are sent along with the node. Master processors insert in the working queue  $q_c$  their own unsolved nodes from  $q_u$ , after these are cleaned up, along with the nodes received from their slaves, and reset all *unsolved* flags; the BorůvkaMSF\* operator is then applied again (see Fig. 7). Note that the pyramidal hierarchy is implemented by merging alternatively subresults across horizontal and vertical subdomain boundaries.

Since no information about the nodes already incorporated in an unsolved tree is passed on to the master processor, multiple edges between remote trees are not altered, as emphasized earlier. Suppose a slave processor has two external edges  $(u_1, v, w_1)$  and  $(u_2, v, w_2)$ .  $u_1, u_2$  are local nodes, while  $v$  is at the master processor.  $u_1$  and  $u_2$  merge into a new component, let this be  $u_1$ , at the slave processor, so that the two external edges connect the same two distinct components rooted at  $u_1$  and  $v$ , respectively. If  $w_1 < w_2$  and  $(u_2, v, w_2)$  is removed, all information received by the master processor about the component  $u_1$  is its root,  $u_1$ , and the set of outgoing edges, among which is  $(u_1, v, w_1)$ . Consequently, when the component to which  $v$

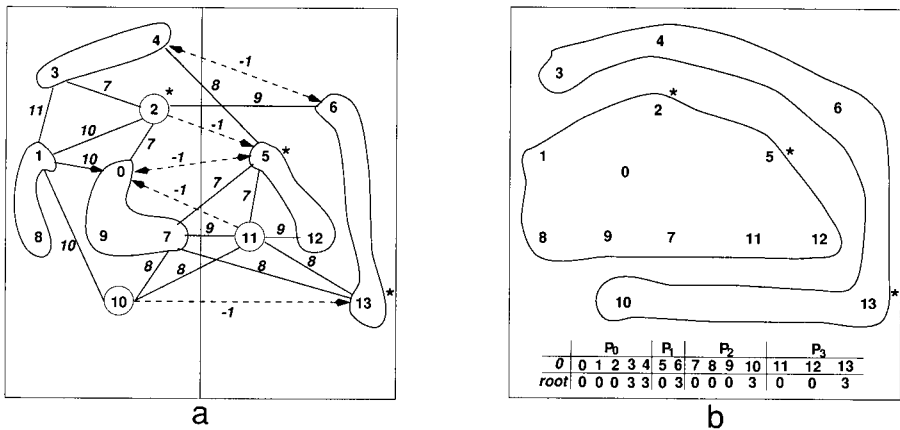


FIG. 8. The result of the BorůvkaMSF\* operator after merging across (a) horizontal and (b) vertical subdomains boundaries and the final labels in the master's root array to be scattered to all processors.

belongs, at the master processor, processes the edge  $(v, u_2, w_2)$ , it will not retrieve the information that  $u_2$  was incorporated into the same component with  $u_1$ . At the master processor,  $root[u_2] = u_2$  and  $u_2$  has an empty edges list. Consequently, it is an external node, and it will be so until the top of the pyramid. Hence, the component containing  $v$  will never become isolated, i.e., solved.

The results of the MSF operator after each merging step are shown in Fig. 8. Let us notice in Fig. 8a that edge  $(5, 6, 8)$  has been removed as an interregion link between two marked components. After the last merging step, there are two global components rooted at 0 and 3, respectively. At the final master, the root array holds incorrect information about all nodes incorporated into trees solved at lower levels of the pyramidal hierarchy. These data have not been passed on, but they still exist in the corresponding processors' root array. Since any time a tree merges with another the new root is less than or equal to the current root, a global MIN reduce operation over all processors' root arrays (see [20]) will set in the final master's root array the missing information. The corresponding ranges of labels in the master's root array, shown in Fig. 8b, are then scattered back to processors and relabeling with the final labels is performed in every subimage (see also [18, 19]).

#### 4. COMPLEXITY ANALYSIS

In this section, a complexity profile of the previously described algorithm is comprised. The complexity of the first stage of the algorithm, local flooding of the catchment basins, is  $T^1 = O((l/\sqrt{P})k + (l^2/P)(1 + kf))$  (see [18]), where  $k, f$  are data-dependent values.

For the second stage of the algorithm, a very accurate analysis is difficult because of the irregular distribution of vertices (basins) and markers to processors. However, in the following, a lower bound estimate of the real performance is given. The analysis relies heavily on the complexity of the local MSF operator for a graph

with  $m$  edges and  $n$  vertices, as presented in [3]. The formula follows easily by observing that after every stage in the local MSF the number of remaining trees is at most half the number of trees at the input of the stage. Indeed, when two trees are merged, one tree emerges in the next stage. If one of the trees has already been enqueued for the next stage, it means that more than two trees in the current stage have contributed to one tree in the following stage. Thus, in stage  $i$  there are at most  $n/2^i$  trees (vertices) and  $(n/2^i)^2$  edges after cleanup. The procedure stops when no trees are left in the working queue, which implies a number of iterations equal to  $\log_2(n)$ . Since selection of the lightest edge is the only time-consuming step, proportional to the number of outgoing edges of a tree, the total complexity is  $O(\sum_{i=0}^{\infty} (n/2^i)^2) = O(n^2)$ . The cleanup after each stage is  $O(m)$  which totals  $O(m \log_2(n))$  for the whole MSF procedure. Thus, the total complexity is  $T_{\text{MSF}}(m, n) = O(\min\{m \log_2(n), n^2\})$ .

In the above analysis, however, the markers were not considered. Active nodes are only nonmarked nodes and marked nodes whose lightest edge is an intraregion link (of weight  $-1$ ). An active node becomes passive (not further processed) if it turns either solved (isolated), or unsolved if the lightest edge to be collapsed emerges toward an external node. The number of unsolved nodes is, however, a fraction of the number of nodes in the surrounding boundary of the current subdomain. The complexity of the first MSF computation in each processor is  $T^{2^0} = T_{\text{comp}}^{2^0} = \max_{0 \leq i < P} \{T_{\text{MSF}}(L_i)\}$ .

At further levels of the pyramidal merging, are active only nodes on each side of the boundary across which combination of the subresults is performed, plus the marked nodes linked by an interregion edge to a nonmarked unsolved node. Thus, if  $L_b$  is the maximum number of unsolved vertices within a boundary subdomain of a processor ( $L_b < l/\sqrt{P}$ ), then in the  $k$ th level of merging there are at most  $O(L_b 2^{\lfloor k/2 \rfloor})$  active nodes and  $O(L_b 2^{\lfloor k/2 \rfloor})$  edges ( $1 \leq k \leq \log_2(P)$ ). The computation complexity at the  $k$ th level is thus  $T_{\text{comp}}^{2^k} = T_{\text{MSF}}(L_b 2^{\lfloor k/2 \rfloor})$ . At each communication step, the unsolved nodes and their outgoing edges are communicated. As emphasized above, the number of unsolved nodes is proportional to the number of regions in the surrounding perimeter of the subdomain. Since a subdomain doubles one of the  $x$  or  $y$  dimension at every merging level, the subdomain perimeter at level  $k$  is  $2(2^{\lfloor k/2 \rfloor} + 2^{\lfloor (k-1)/2 \rfloor}) \times l/\sqrt{P}$ . The communication complexity is then  $T_{\text{comm}}^{2^k} = O((2^{\lfloor k/2 \rfloor} + 2^{\lfloor (k-1)/2 \rfloor}) L_b)$  (the information about all unsolved nodes is first packed and then a single message is sent once). An additional computational complexity is due to packing and unpacking the message,  $O((2^{\lfloor k/2 \rfloor} + 2^{\lfloor (k-1)/2 \rfloor}) L_b)$ . The computational complexity of the  $k$ th stage becomes thus  $T_{\text{comp}}^{2^k} = T_{\text{MSF}}(L_b 2^{\lfloor k/2 \rfloor}) + O((2^{\lfloor k/2 \rfloor} + 2^{\lfloor (k-1)/2 \rfloor}) L_b)$ . The total complexity of the  $k$ th merging level is  $T^{2^k} = T_{\text{comp}}^{2^k} + T_{\text{comm}}^{2^k}$ ,  $1 \leq k \leq \log_2(P)$ . Finally, the total complexity of the second stage is  $T^2 = \sum_{k=0}^{\log_2(P)} T^{2^k}$  and of the whole algorithm  $T = T^1 + T^2$ .

As a remark, it should be noticed that the complexity depends loosely on the number of marked nodes. The analysis depends, however, on the image data: first, on the number of regions existent in the image, as it turns out from the analysis of  $T^{2^0}$ , and on the plateaus topology and size, which affect the  $k, f$  values in the  $T^1$  time component; and second, through the number  $L_b$  of unsolved vertices remaining after the first application of the MSF operator. This number is very low

for an image with few regions in the neighborhood of a subdomain boundary, or with markers distributed evenly to catchment basins on each side of the subdomain boundary (interregion links between marked regions are invalid, and consequently discarded).

Moreover, the larger the number of processors, the smaller the subdomain area and hence, smaller  $T^1$  and  $T^2_0$ . But, the number of merging levels increases with larger number of processors, and also both  $T^2_{\text{comp}}$  and  $T^2_{\text{comm}}$  become more important. Consequently, good performance is obtained as long as the overhead due to merging of the MSF subresults do not overtake the gain of the local flooding and MSF operator at the 0th level of merging.

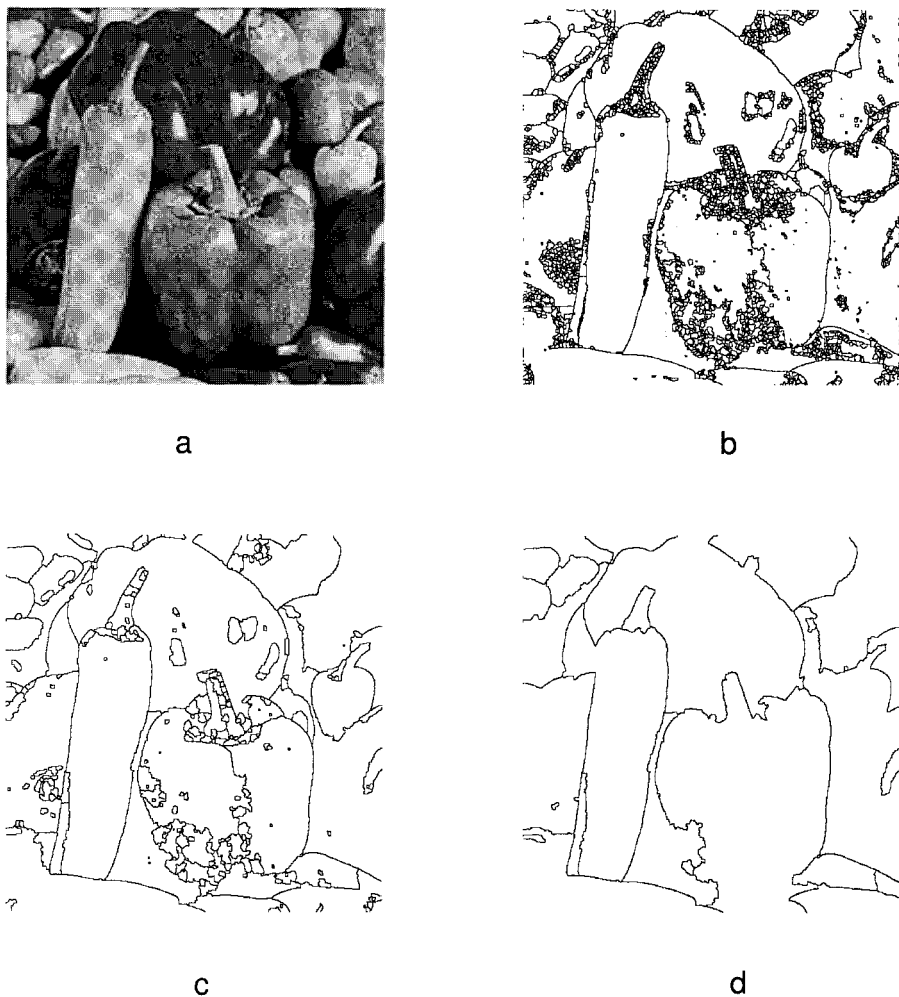
## 5. EXPERIMENTAL RESULTS

Results of the algorithm implementation using *Message Passing Interface* (MPI) on the Cray T3D parallel computer are presents in this section. The behavior of the algorithm was studied on images of different sizes ( $256 \times 256$ ,  $512 \times 512$ ,  $1024 \times 1024$ ), but also on equal-sized images ( $512 \times 512$ ) with different complexities or with different sets of markers. Marker images were obtained by collecting the regional maxima of the original image smoothed by a multiscale dilation with scale  $\sigma = 2.0$  (see [8]) or manually, by pointing different pixel locations inside regions of interest.

Execution times ( $T$ ) were measured for all test images excluding data loading, distribution, retrieval, and saving. Timings were collected for the two stages of the algorithm, local flooding of the catchment basins ( $T^1$ ) and marker-based region merging with the Borůvka-like MSF operator ( $T^2$ ). From the latter, a separate component ( $T^2_0$ ), namely the time spent by the MSF operator at level 0 (when the first partial results of the MSF are obtained), was also measured. Furthermore, the number of catchment basins in the oversegmented image ( $\# \text{regs}_o$ ) is compared against the number of marked regions ( $\# \text{regs}_m$ ). All these values are tabulated for three images in Table 1 when using up to 128 processors. In a separate table,

**TABLE 1**  
**Execution Times (in s)**

Image	<i>Cermet</i> ( $256 \times 256$ )				<i>Lenna</i> ( $512 \times 512$ )				<i>People</i> ( $1024 \times 1024$ )			
$\# \text{regs}_o$	418				5370				14684			
$\# \text{regs}_m$	156				709				1255			
$P$	$T^1$	$T^2_0$	$T^2$	$T$	$T^1$	$T^2_0$	$T^2$	$T$	$T^1$	$T^2_0$	$T^2$	$T$
1	0.730	0.0246	0.024	0.913	2.861	1.002	1.002	8.062	12.628	3.941	3.941	28.745
2	0.403	0.0123	0.033	0.539	1.848	0.467	1.772	5.144	8.763	1.838	2.250	20.076
4	0.211	0.0061	0.069	0.336	0.899	0.227	0.830	3.102	4.379	0.781	2.232	10.524
8	0.121	0.0032	0.079	0.243	0.485	0.109	0.578	1.699	2.257	0.374	3.065	6.874
16	0.067	0.0016	0.095	0.190	0.282	0.052	0.557	1.280	1.066	0.173	1.230	4.067
32	0.040	0.0009	0.098	0.171	0.144	0.023	0.439	0.825	0.605	0.074	1.517	2.983
64	0.029	0.0005	0.123	0.155	0.084	0.011	0.416	0.628	0.312	0.034	0.706	1.656
128	0.028	0.0003	0.121	0.154	0.048	0.005	0.378	0.568	0.160	0.016	0.727	1.144



**FIG. 9.** *Peppers* (a) original and (b) oversegmented (2795 regs) image and marker-based segmented image with markers collected (c) by multiscale morphology (264 regs) and (d) manually (27 regs).

Table 2, the same type of results are gathered for test image *Peppers* (see Fig. 9) ( $512 \times 512$ ), but using different sets of markers.

As emphasized in the previous section,  $T^1$  and  $T^{2_0}$  scale very well with increasing number of processors in every test image case, while  $T^2$  has a data-dependent evolution. Another observation is that the number of markers does not have a strong impact upon the running time.  $T^{2_0}$  is slightly larger for a small number of markers. Indeed, the number of nonmarked nodes is larger too, and hence there are more active vertices in the MSF computation than for the case of a large number of marked nodes. On the other hand, a small number of markers implies a small number of unsolved marked nodes, i.e., a small amount of data to be communicated.

TABLE 2

Execution Times for Image *Peppers* with Different Marker Sets

Image	<i>Peppers</i> (512 × 512)					<i>Peppers</i> (512 × 512)				
	$T^1$	$T^{2_0}$	$T^2$	$T$	$\frac{T(1)}{P \times T(P)}$	$T^1$	$T^{2_0}$	$T^2$	$T$	$\frac{T(1)}{P \times T(P)}$
#regs <sub>o</sub>	2795					2795				
#regs <sub>m</sub>	264					27				
1	2.775	0.523	0.523	5.444	1.000	2.778	0.639	0.639	5.608	1.000
2	1.713	0.236	0.465	3.163	0.860	1.713	0.284	0.574	3.464	0.809
4	0.866	0.112	0.454	1.732	0.786	0.866	0.128	0.451	1.766	0.794
8	0.468	0.052	0.514	1.111	0.612	0.469	0.060	0.494	1.112	0.630
16	0.254	0.025	0.387	0.714	0.476	0.254	0.029	0.400	0.731	0.479
32	0.138	0.011	0.359	0.590	0.288	0.138	0.013	0.377	0.588	0.298
64	0.075	0.005	0.335	0.439	0.193	0.075	0.006	0.346	0.444	0.197
128	0.050	0.002	0.301	0.399	0.106	0.053	0.002	0.319	0.433	0.101

The relative speedup given by the ratio  $SP(P) = T(1)/T(P)$  is also illustrated in Fig. 10. The speedup increases with the number of processors for all images and it saturates around 64 processors. Of course, better performance is obtained with large images.

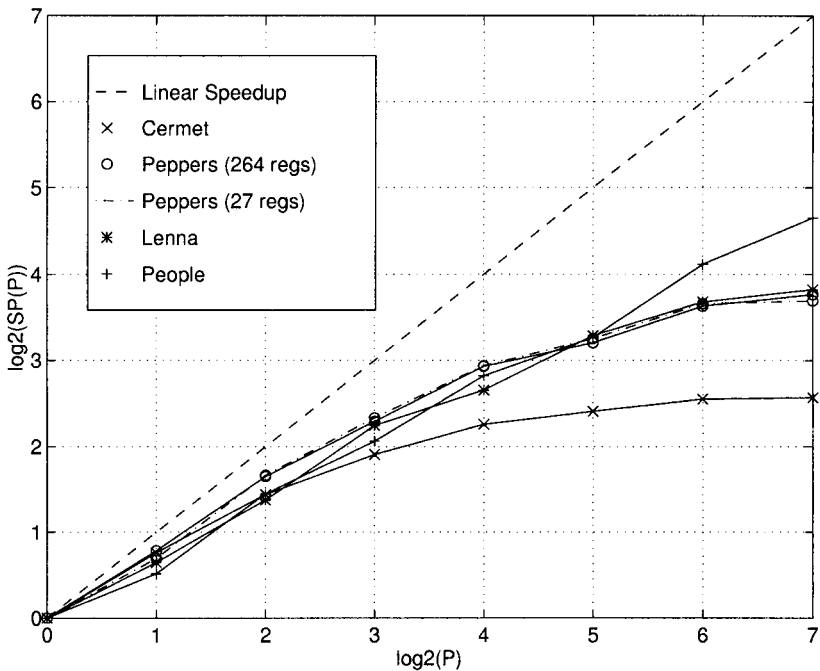


FIG. 10. Relative speedup versus number of processors.

## 6. CONCLUSIONS

In this paper, we have described a parallel marker-based watershed algorithm for image segmentation. The algorithm offers an efficient method for computing in a distributed fashion the catchment basins of a gray-scale image and, at the same time, for merging nonmarked regions to marked ones, using a constrained Borůvka-like MSF operator. In both stages, locality and concurrency were best exploited, while the amount of communication was reduced to the needed data set.

A theoretical analysis of the main modules in the parallel algorithm was included in the paper to predict the behavior of the algorithm and interpret the experimental results. The latter were obtained with an MPI implementation on a Cray T3D parallel computer. Good performance of the algorithm has been observed for large images from both running time and relative speedup viewpoints. Additionally, the latter property emphasizes an ascending scalability of the algorithm, up to 64 processors. Moreover, the oversegmentation was significantly reduced.

## REFERENCES

1. H. Alnuweiri and V. Prasanna, Parallel architectures and algorithms for image component labeling, *IEEE Trans. Pattern Anal. Mach. Intell.* **14**, 10 (Oct. 1992), 1014–1034.
2. S. Beucher and F. Meyer, The morphological approach to segmentation: The watershed transformation, in “Mathematical Morphology in Image Processing” (E. R. Dougherty, Ed.), pp. 433–481, Dekker, New York, 1993.
3. D. Cheriton and R. E. Tarjan, Finding minimum spanning trees, *SIAM J. Comput.* **5**, 4 (Dec. 1976), 724–742.
4. T. Cormen, C. Leiserson, and R. Rivest, “Introduction to Algorithms,” MIT Press, Cambridge, MA, 1990.
5. A. Gibbson and W. Rytter, “Efficient Parallel Algorithms,” Cambridge Univ. Press, Cambridge, UK, 1988.
6. R. L. Graham and P. Hell, On the history of the minimum spanning tree problem, *Ann. History Comput.* **7**, 1 (Jan. 1985), 43–57.
7. J. JáJá, “An Introduction to Parallel Algorithms,” Addison–Wesley, Reading, MA, 1992.
8. P. Jackway, Gradient watersheds in morphological scale-space, *IEEE Trans. Image Process.* **5**, 6 (June 1996), 913–921.
9. V. Kumar, A. Grama, A. Gupta, and G. Karypis, “Introduction to Parallel Computing—Design and Analysis of Algorithms,” Benjamin–Cummings, Redwood City, CA, 1994.
10. F. Meyer, Topographic distance and watershed lines, *Signal Process.* **38**, 1 (July 1994), 113–125.
11. F. Meyer and S. Beucher, Morphological segmentation, *J. Visual Comm. Image Rep.* **1**, 1 (Sept. 1990), 21–46.
12. F. Meyer, Minimum spanning forests for morphological segmentation, in “Mathematical Morphology and Its Applications to Image Processing,” pp. 77–84, Kluwer Academic, Dordrecht, 1994.
13. A. N. Moga, T. Viero, M. Gabbouj, M. Nölle, G. Schreiber, and H. Burkhardt, Parallel watershed algorithm based on sequential scanlines, in “Proc. IEEE Workshop on Nonlinear Signal and Image Processing, Neos Marmaras, Halkidiki, June 1995,” Vol. II, pp. 991–994.
14. A. Moga and M. Gabbouj, A parallel watershed algorithm based on the shortest paths computation, in “Parallel Programming and Applications,” pp. 316–324, IOS Press, Amsterdam, May 1995.
15. A. N. Moga, B. Cramariuc, and M. Gabbouj, A parallel watershed algorithm based on rainfalling simulation, in “Proc. European Conference on Circuit Theory and Design, Istanbul, Turkey, August 1995,” Vol. 1, pp. 339–342.

16. A. Moga, B. Cramariuc, and M. Gabbouj, An efficient watershed segmentation algorithm suitable for parallel implementation, in "Proc. IEEE International Conference on Image Processing, Washington, D.C., October 1995," Vol. II, pp. 101–104.
17. A. N. Moga and M. Gabbouj, A parallel marker based watershed transformation, in "Proceedings IEEE International Conference on Image Processing, Lausanne, Switzerland, September 1996," Vol. II, pp. 137–140.
18. A. N. Moga and M. Gabbouj, Parallel image component labeling with watershed transformation, *IEEE Trans. Pattern Anal. Mach. Intell.* **19**, 5 (May 1997), 441–450.
19. A. N. Moga, "Parallel Watershed Algorithms for Image Segmentation," Ph.D. thesis, Tampere University of Technology, Tampere, Finland, February 1997.
20. Message Passing Interface Forum, MPI: A message-passing interface standard, Technical Report University of Tennessee, Knoxville, Tennessee, June 1995, Version 1.1.
21. M. J. Quinn, "Designing Efficient Algorithms for Parallel Computers," McGraw–Hill, New York, 1987.
22. R. Tarjan, "Data Structures and Network Algorithms," Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.